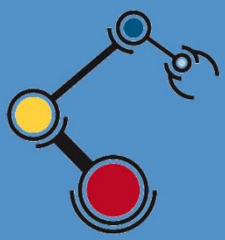
Escuela Politécnica Superior

Control de velocidad de un motor AC mediante control de fase con Arduino



Máster Universitario en Automática y Robótica

Trabajo Fin de Máster

Autor:

Jesús Mariano Moreno Tendero

Tutor/es:

Francisco A. Candelas Herías

Universitat d'Alacant
Universidad de Alicante

Julio 2018

RESUMEN

Este proyecto consiste en realizar un control de velocidad de un pequeño motor AC monofásico de 230V, como puede ser el de un taladro, mediante un Arduino, de forma que sea posible establecer en cualquier momento la velocidad a la que se desea que gire el motor, y el programa del Arduino garantice que el motor mantiene esa velocidad independientemente de la carga a la que se vea sometido el motor, dentro de unos límites de trabajo. Para actuar sobre el motor se realizará un control de fase. En este punto se puede considerar un algoritmo de control digital PID genérico, y ajustar sus parámetros. La aplicación incluirá además una pequeña interfaz de usuario que permita a una persona encender y apagar el motor, así como establecer y monitorizar su velocidad. Para esto se implementará un protocolo de comunicación que permita operar el Arduino desde una aplicación gráfica en un PC.

SUMARY

This project is to perform a speed control of a small single fhase AC motor of 230 volts, such as a drill, using an Arduino, so that it is possible to set the motor speed at any time, and the program of Arduino guarantees that the motor maintains that speed independently of the load to which the motor is subjected, within working limits. To control the motor, phase control will be carried out. At this point you can consider a generic PID digital control algorithm and adjust its parameters. The application will also include a small user interface that allows a person to turn the engine on and off, as well as establish and monitor its speed. For this, a communication protocol will be implemented that allows operating the Arduino from a graphic application on a PC.

RESUM

El projecte consisteix en realitzar el control de velocitat d'un xicotet motor AC monofàsic de 230V, com podria ser el d'un taladro, mitjançant un Arduino, de manera que sigui possible establir en qualsevol moment la velocitat a la que es desitja que giri el motor, amb el programa d' Arduino garantirà que el motor mantega aquesta velocitat independentment de la carrega a la estigi sotmès. Per actuar sobre el motor es realitzarà un control de fase. En aquest punt, es pot considerar un algoritme de control PID genèric, amb ajustar els seus paràmetres. La aplicació inclourà a mes una interfície d'usuari que permetrà encendre i apagar el motor, ací com establir y monitoritzar la seva velocitat. Per a fer açò es implementarà un protocol de comunicacions que permetrà operar el Arduino des d'una aplicació gràfica en un PC.

PALABRAS CLAVE

Arduino, Control de velocidad, Control de fase, Motor AC, Control PID.

KEYWORDS

Arduino, Speed control, Phase control, AC Motor, PID Control.

PARAULES CLAU

Arduino, Control de velocitat, Control de fase, Motor AC, Control PID.

JUSTIFICACIÓN Y OBJETIVOS

Los principales motivos que han llevado a la realización de este trabajo final de máster consisten en el interés por conocer mejor los sistemas de control automáticos y como llevarlos a cabo. Durante el máster se han realizado tareas sobre este tema, pero principalmente mediante simulación, lo cual ha despertado cierta curiosidad en mi sobre este tema y ganas de llevar a cabo físicamente los conocimientos adquiridos sobre la materia. Además, tengo interés en profundizar en temas vistos en el máster de programación y HMI.

Los objetivos generales son realizar el control de velocidad y monitorización de un pequeño motor AC monofásico actuando sobre el con un circuito de control de fase mediante el empleo de un Arduino. Normalmente el control de fase se suele realizar mediante un mando manual, o controlado mediante un “micro” en bucle abierto, pero en el caso de este proyecto se realizará en bucle cerrado.

INDICE DE CONTENIDOS

INDICE DE FIGURAS

INDICE DE TABLAS

1. INTRODUCCIÓN

1.1. VISIÓN GENERAL

Hoy en día es necesario el control de velocidad de motores eléctricos para diversas aplicaciones, como por ejemplo en juguetes, aviación, la automoción, diversas herramientas como tornos o fresadoras. Para controlar estos motores es preciso emplear reguladores y la teoría de control.

Los reguladores permiten respuestas rápidas y precisas ante cambios en el sistema, lo cual viene muy bien a la hora de controlar motores eléctricos, ya que podemos mantener en todo momento la velocidad de este a pesar de que aumente o disminuya el par resistente de este.

El control digital es la implementación de un regulador mediante un algoritmo que se ejecuta en un computador. Este regulador proporciona las señales de control necesarias para que el sistema tenga un comportamiento determinado.

Para poder realizar dicho control el regulador debe conocer en todo momento el objetivo a cumplir (consigna o referencia) y también ha de tener “feedback” para poder saber el estado actual de la variable a controlar del sistema. A esto se le conoce por control en bucle cerrado (figura 1).

Para implementar el control del sistema que se pretende realizar, se empleará cómo controlador un microcontrolador Arduino, en concreto un Arduino MEGA 2560 (figura 2) y como sistema o proceso, un motor de corriente alterna monofásico con un sensor óptico a modo de encoder. La consigna, en este caso, es una velocidad que se quiere alcanzar.

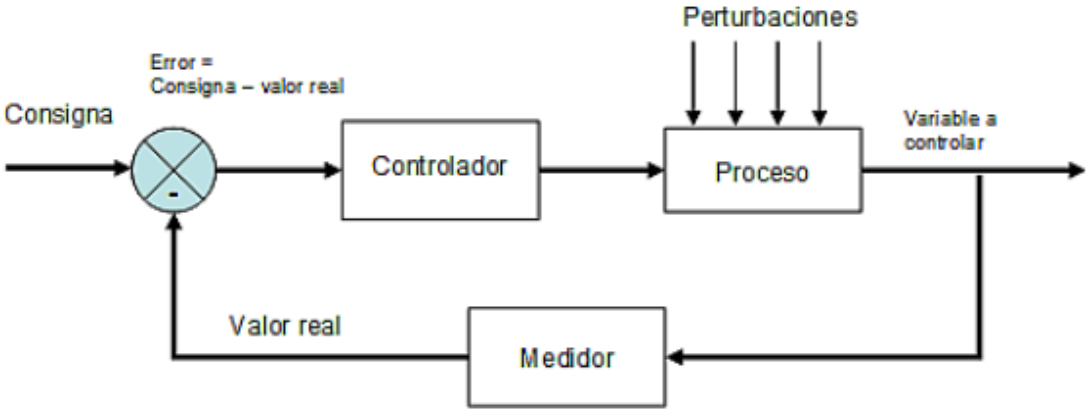


Figura 1: Sistema de control en bucle cerrado.

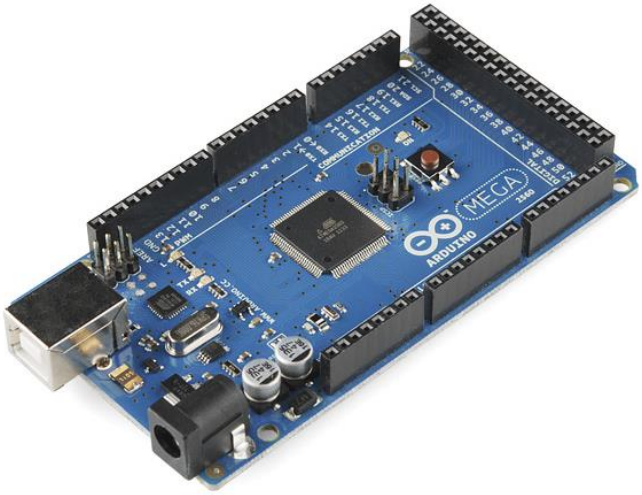


Figura 2: Arduino MEGA 2560.

1.2. ORGANIZACIÓN DE LOS CONTENIDOS

En el **capítulo 1**, se realiza una introducción de las tareas a desarrolladas en este trabajo final de máster. En el **capítulo 2**, se hablan de trabajos relacionados y se explican conceptos básicos sobre reguladores PID, el funcionamiento y control de un motor monofásico síncrono y como medir su velocidad, y también se explica que es el control de

fase. En el **capítulo 3**, se explican los objetivos concretos de este TFM. En el **capítulo 4**, se muestra la metodología empleada para el desarrollo de este trabajo. Posteriormente en el **capítulo 5** se muestra el desarrollo del trabajo, (cálculos, esquemas, código de programación). En el **capítulo 6** se detallan pruebas realizadas, en el **capítulo 7** se encuentra un presupuesto. En el **capítulo 8** las conclusiones, y en el **capítulo 9** las referencias consultadas para la elaboración del trabajo.

Para finalizar se han añadido **anexos** en los que se puede ver el código de programación y los esquemas electrónicos.

2. ESTADO DEL ARTE

La aplicación de reguladores PID en Arduino no es novedad, ya que se suelen emplear para controlar motores de corriente continua [4] como por ejemplo el control de velocidad de ventiladores [5], incluso existen librerías descargables dedicadas al PID [6]. También existen ya aplicaciones de control de fase implementadas con Arduino, por ejemplo, en [7] y [8] emplea esta técnica para hacer un "dimer" que controla la luminosidad de una lampara, o por ejemplo en [9] se emplea el control de fase para calentar agua con el exceso de energía producido por unas placas solares. Pero a pesar de que estas técnicas están bastante introducidas, no se encuentra que se emplee el control PID en Arduino para el control de velocidad de un motor de corriente alterna monofásico por medio del control de fase.

2.1. REGULADOR PID

Un regulador PID (proporcional, integral, derivativo) es un algoritmo de control retroalimentado (bucle cerrado) que calcula un error entre la consigna y el valor de la variable medida, para en función de este error enviar una señal de control al amplificador que opera el accionador, en este caso un motor, con el objetivo de eliminar dicho error en régimen permanente, dicho de otra manera, se encarga de que el valor medido sea igual a la consigna al cabo de un tiempo deseado.

Un regulador PID se compone de tres acciones de control:

$$U = P + I + D$$

Algunas aplicaciones no requieren el uso todos los parámetros del PID para proporcionar un control apropiado al sistema, para no usar un parámetro simplemente hay que asignar un valor de cero a la ganancia del parámetro que no se va a emplear. Dependiendo de los parámetros activos, los reguladores se pueden llamar PI, PD, P o I. La

parte PI trata de eliminar el error en régimen estacionario, mientras que la parte PD pretende conseguir una respuesta rápida y reducir las oscilaciones en los transitorios.

2.1.1. ACCIÓN DE CONTROL PROPORCIONAL (P)

El término proporcional del PID genera una salida proporcionalmente con el error actual del sistema donde. Esta respuesta se puede ajustar como una constante K_p multiplicando al error. Esta ganancia se conoce como ganancia proporcional.

$$P = K_p \cdot e_{(t)}$$

Cuanto mayor sea el error mayor será la señal de control y más rápida será la respuesta del sistema.

Por si solo el término proporcional no siempre es capaz de eliminar del todo el error en régimen estacionario. El error que persiste cuando el sistema está en régimen permanente se denomina offset. Para suplir este offset es conveniente añadir el término integral.

2.1.2. ACCIÓN DE CONTROL INTEGRAL (I)

El término integral es proporcional a la magnitud del error y a la duración de este. O dicho de otra manera, es la suma de todos los errores en cada instante de tiempo (integración de los errores). Con esta acción se debería corregir el offset mencionado anteriormente.

El error acumulado se multiplica por la ganancia integral K_i que se encarga de dar mayor o menor importancia a esta acción de control.

$$I = K_i \int_0^t e_{(t)} dt$$

Combinando esta acción con la acción proporcional se acelera la respuesta del sistema obteniendo un menor tiempo de respuesta. Además, obtenemos la ventaja de eliminar el error en régimen estacionario. Por otra parte, tenemos la desventaja de que esta acción puede crear oscilaciones alrededor del valor de la consigna, esto es debido a que tiene en cuenta todos los errores acumulados anteriormente.

En la práctica hay que tener en cuenta la necesidad de limitar el valor de la suma, en máximo positivo y mínimo negativo, porque la salida de un controlador real dispone de limitaciones físicas según su construcción.

2.1.3. ACCIÓN DE CONTROL DIFERENCIAL (D)

La acción derivativa calcula la variación del error mediante la pendiente del error en cada instante de tiempo, o lo que es lo mismo, la derivada con respecto al tiempo del error, y multiplica este valor con la ganancia derivativa $K_{d,}$ la cual indica cuanto peso tiene esta acción con respecto a toda la acción de control.

$$D = K_d \frac{d}{dt} e_{(t)}$$

Esta acción derivativa ralentiza la dinámica del regulador y cuanto menor es el error, es aún más lento. Esto puede suponer una ventaja, ya que con ella se pueden reducir las oscilaciones creadas por la acción integral y las perturbaciones, haciendo el control más estable. En contraposición esta acción es muy sensible a interferencias ya que las amplifica pudiendo llevar al sistema a la inestabilidad.

2.1.4. ECUACIÓN DEL REGULADOR

Llamando u(t) a la acción de control completa la ecuación que define el algoritmo representado en la figura 3 es la siguiente:

$$u_{(t)} = K_p \cdot e_{(t)} + K_i \int_0^t e_{(t)} dt + K_d \frac{d}{dt} e_{(t)}$$

Llegados a este punto hay que tener en cuenta varias consecuencias. Cuanto mayor es la ganancia proporcional, mayor es la rapidez del sistema, pero valores muy grandes pueden ocasionar oscilaciones o incluso la inestabilidad. Cuanto mayor es la ganancia integral más rápido se eliminan los errores, pero puede provocar sobreoscilación del sistema. Cuanto mayor es la ganancia del sistema mas se reducen las oscilaciones, pero más lento es el sistema.

Del párrafo anterior se puede deducir que las ganancias integral y derivativa son antagonistas, ya que la integral hace al sistema rápido y la derivativa lo hace lento. Esto puede llevarnos a pensar que son incompatibles ya que se anularían, pero afortunadamente no es así, solo hay que llegar a un equilibrio entre ellas.

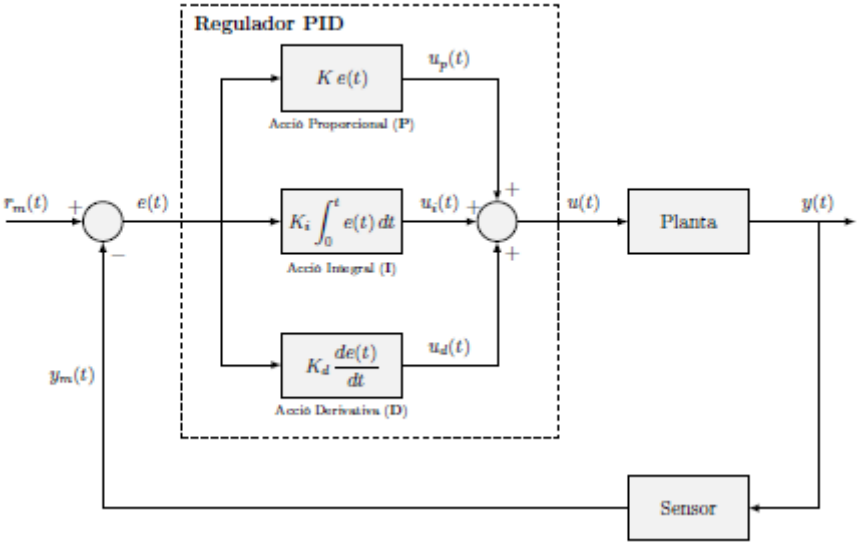


Figura 3: Diagrama de bloques de un control PID completo.

2.1.5. ECUACIÓN DEL REGULADOR DISCRETA

Como un microcontrolador solo es capaz de procesar valores discretos y finitos, es necesario discretizar la ecuación del regulador PID. Una vez discretizada se ha de calcular

la acción de control para cada instante de tiempo muestreado. Para cada instante de tiempo se vuelve a calcular la acción de control empleando la nueva salida del sistema.

Llamando T_S al periodo de muestreo, o tiempo entre medidas, en este caso la integral de la acción de control se puede aproximar como el sumatorio de los errores en el tiempo por el tiempo de medida $\sum_{k=0}^{t}(T_S \cdot e_k)$ y la derivada se puede aproximar al cociente incremental, y empleando la diferencia hacia atrás del error divididos por el tiempo de muestreo ($\frac{e_{(t)}-e_{(t-1)}}{T_S}$). Existen métodos mejores de aproximación, como la integral trapezoidal, u otros métodos de diferenciación, pero los anteriores son los más usados en microcontroladores pequeños.

Según [1], la ecuación discreta del regulador se puede expresar de la siguiente manera:

$$u_{(t)} = K_p \cdot e_{(t)} + K_i \cdot T_S \sum_{k=0}^{t} e_k + K_d \frac{e_{(t)} - e_{(t-1)}}{T_S}$$

En la ecuación anterior $e_{(t)}$ es el error del sistema en el instante t, T_S es el periodo de muestreo de la señal y $K_p, K_i \; y \; K_d$ son las ganancias proporcional, integral y derivativa, respectivamente. En el Código 1 se puede ver un ejemplo de implementación de un regulador PID discretizado.

```
  //* Errores *//
  error = consigna - velocidad;
  errAcum += (error * dT/1000);
  errDif = (error - errAnterior)*1000 / dT; // dT en ms; dT/1000 = segundos

  //* PID *//
    P = kp * error;
    I = ki * errAcum;
    D = (float)(kd * errDif*1000 / dT );
    u = P + I + D;

 //* Actualización de variables para la proxima iteración
  errAnterior = error;
```

Código 1: Ejemplo de implementación del algoritmo PID discretizado.

2.2. MOTOR DE CORRIENTE ALTERNA MONOFÁSICO

El empleo de motores monofásicos está muy extendido, estos se suelen emplear en campos como herramientas y electrodomésticos pequeños. **En este caso se expondrá el motor monofásico síncrono o motor universal únicamente ya que es el accionamiento que se pretende controlar con este proyecto.**

Controlar la velocidad de estos motores se puede realizar de dos formas, variando la tensión de alimentación y variando la frecuencia de alimentación. En este trabajo la variación de la velocidad se realizará variando únicamente la tensión de alimentación y de entre todas las formas posible de variar la tensión, se ha escogido la del control de fase, que consiste en variar la tensión eficaz de alimentación (V_{RMS}) como se verá más detalladamente en el punto 3.2.

2.2.1. CONSTITUCIÓN Y FUNCIONAMIENTO

Este tipo de motor dispone de las siguientes partes constructoras (figura4):

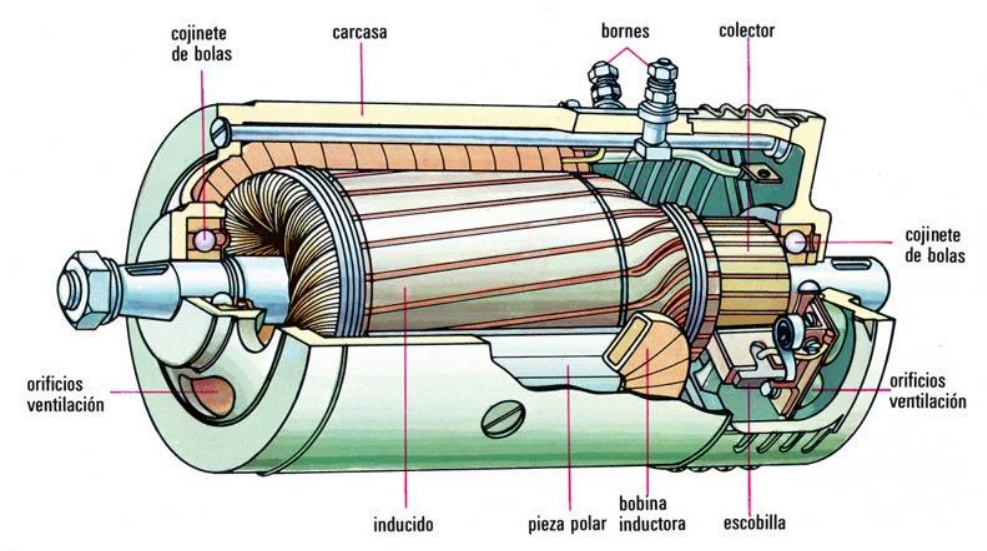


Figura 4: Motor universal.

Como menciona [2] en su capítulo 5, el bobinado inductor y el bobinado inducido están conectados en serie a través de las escobillas y el colector. Al ser recorridos por una corriente, el bobinado inductor forma el campo magnético y el inducido por la ley de Laplace, al ser recorrido por la corriente y sometido a la influencia del campo magnético inductor, se desplaza, dando origen al giro del rotor. Si aumenta el campo aumenta la fuerza dependiendo de los parámetros dinámicos de la carga, aumentará también la velocidad.

Una particularidad de este motor es que es capaz de funcionar con corriente continua y con corriente alterna.

2.2.2. CONTROL DE FASE

Este método se basa en emplear un circuito de un convertidor AC/AC, el cual funciona manipulando la onda sinusoidal que se entrega al accionamiento mediante el control del ángulo en que se dispara de un interruptor electrónico, como un SCR o TRIAC, en serie con el accionamiento, de forma que se varía así la potencia eficaz o media que se proporciona a dicho accionamiento.

Un SCR o rectificador controlado de silicio, es un tipo de tiristor que posee tres conexiones, ánodo (A), cátodo (C) y puerta (G) como se puede comprobar en la figura 5 a. Cuando circula una corriente por su terminal llamado puerta, este permite la conducción de corriente desde el ánodo al cátodo, hasta que se extingue la corriente de ánodo-cátodo. Si colocamos dos SCR en antiparalelo y unimos sus terminales “puerta”, obtenemos un Triac (figura 5 b) con la ventaja de poder realizar la conducción en ambos sentidos.

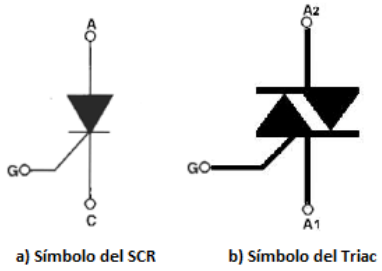


Figura 5: Símbolos del SCR (a) y Triac (b).

La siguiente ecuación, obtenida de [3] representa la tensión eficaz en bornas (V_o) del motor en función del ángulo de disparo (α) del TRIAC, siendo V_s la tensión eficaz de alimentación.

$$V_o = \sqrt{\frac{1}{\pi}\int_{\propto}^{\pi} 2 \cdot V_s^2 \cdot sen^2(\omega t) d(\omega t)} = V_s \cdot \sqrt{1 - \frac{\propto}{\pi} + \frac{sen\,(2 \cdot \propto)}{2\pi}}$$

En este caso el ángulo de disparo es un valor de 0º a 180º correspondiente a la función sinusoidal de la onda dentro de un semiperiodo de la onda, y por lo tanto se corresponde con un valor de tiempo de disparo de 0 a T/2, siendo T el periodo.

Para este control, como se puede ver en la figura 6, se requiere esperar un tiempo desde el comienzo de la onda hasta cuando decidimos activar los tiristores, y esto dependerá del ángulo (a su vez potencia) que se requiera fijar para el funcionamiento. En este trabajo se modificará este parámetro en función de la velocidad a la que se quiera que funcione el motor.

Para poder realizar este tipo de control es necesario disparar el TRIAC de forma sincronizada con la onda sinusoidal de la red eléctrica, para ello es necesario detectar cuando la onda de tensión de la red realiza su cruce por cero (cuando la señal tiene cero voltios por valor). Una vez detectado el cruce por cero, se debe esperar un tiempo (ángulo de disparo) para disparar el TRIAC y que este “cierre el circuito” activando el motor. En este caso, como la alimentación es de 230V a 50Hz de la red eléctrica europea, el tiempo de retardo deberá ser de entre 0 y 10ms que corresponden al 100% y 0% de la potencia eficaz entregada al motor respectivamente, como se puede observar en la figura 6.

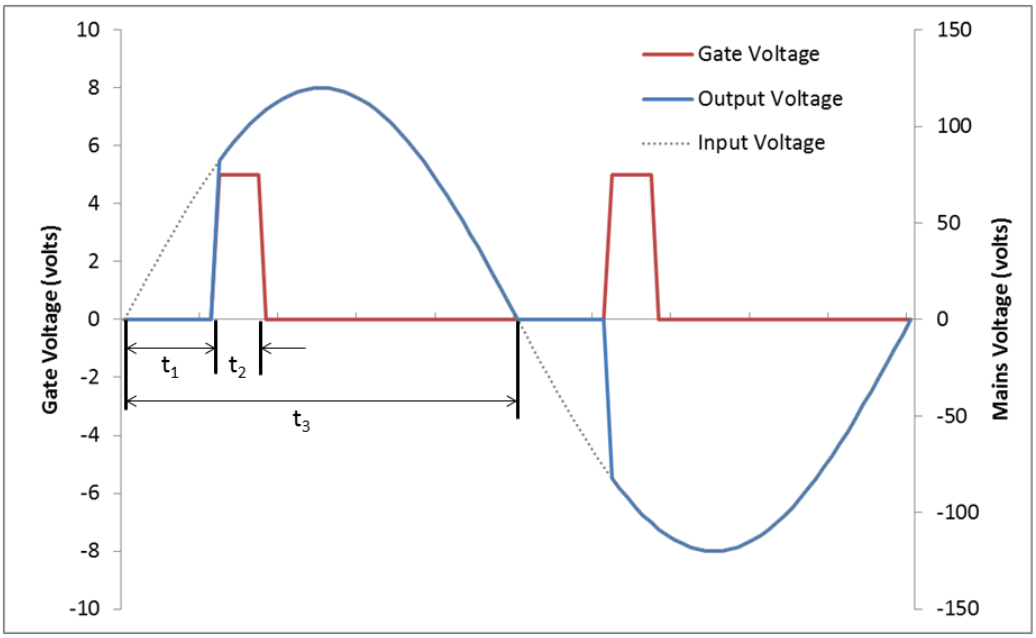


Figura 6: Ondas del control de fase, t1 es el retardo de disparo del TRIAC, t2 es el tiempo de conducción del TRIAC y t3 es el tiempo de un semiperiodo de la onda original [9].

2.2.3. MEDICIÓN DE LA VELOCIDAD DE UN MOTOR

Para medir la velocidad, de los motores se suelen emplear encoders incrementales. Un encoder incremental genera pulsos según rota el eje del motor y estos pueden generar desde un pulso por vuelta o "paso por vuelta", hasta muchos pulsos por vuelta.

La manera más común de generar los pulsos es mediante un disco codificado con ranuras y enganchado al eje de tal manera que el disco gira con el eje. Un sensor óptico detecta el paso de las ranuras (figura 7) y de esta forma sabiendo cuantas ranuras tiene el disco se puede deducir cuantas ranuras son una vuelta, y conociendo el número de vueltas por unidad de tiempo se puede deducir la velocidad.

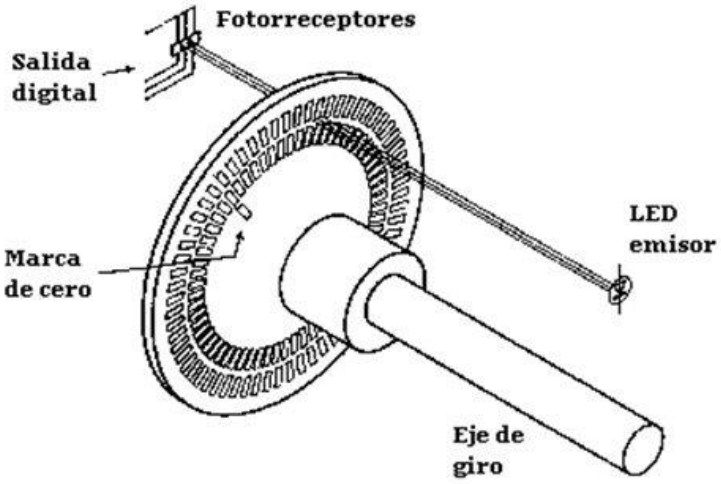


Figura 7: Encoder incremental.

En el caso que nos atañe, el motor tiene instalado un pequeño sensor óptico colocado en una parte fija del motor, y una banda reflectante colocada en una parte giratoria. De esta manera el sensor detectará en cada vuelta la banda reflectante como si fuese un encoder de un paso por vuelta.

3. OBJETIVOS A DESARROLLAR

El objetivo principal de este trabajo es el control de velocidad de un pequeño motor AC monofásico de 230V de un taladro Dremel® 300 independientemente de las perturbaciones debidas al cambio de la carga del motor, que dependerá de momento del taladrado, según el material principalmente. Este control se implementará en un Arduino Mega 2560. Para conseguirlo se han de conseguir los siguientes objetivos:

- Se ha de conseguir controlar el disparo del Triac desde el Arduino para poder llevar a cabo el control de fase.
- Se requiere sincronizar el disparo del triac con el paso por cero de la red eléctrica.
- Se ha de conseguir medir la velocidad del motor con el sensor que lleva incorporado.
- Es necesario implementar un algoritmo de control PID en Arduino para así mantener la velocidad establecida en función de la medida.
- Se ha de determinar el tiempo de disparo del Triac en función de la señal del control de salida del regulador PID.
- Diseñar una aplicación HMI para monitorizar el sistema y mandar la referencia desde el PC.
- Establecer una comunicación serie entre el Arduino y la aplicación HMI para hacerla funcional.

4. METODOLOGIA

Para llevar a cabo este proyecto es necesario no intentar abordar todo el problema de golpe porque sería demasiado complejo corregir posibles errores, por ello se ha dividido el sistema a desarrollar en varias partes.

Primeramente, se ha calculado e implementado los circuitos de detección de paso por cero, lectura del encoder, y el circuito del controlador de fase. Una vez implementados los circuitos se ha comprobado su funcionamiento con la ayuda de un osciloscopio y generador de funciones.

Después se ha procedido, por separado, a realizar el código de programación en Arduino IDE para la lectura del encoder y cálculo de la velocidad, detección de paso por cero, el regulador PID, y el código para el disparador del Triac. Posteriormente se han comprobado los códigos, una vez más con la ayuda del osciloscopio y generador de funciones.

Cuando se ha comprobado que los códigos independientes funcionan correctamente, se han juntado y acondicionado para crear una sinergia entre ellos. Posteriormente se ha realizado el código para la comunicación con la aplicación del PC.

Paralelamente al código implementado en Arduino, se ha realizado la aplicación HMI y comunicación con Arduino con la ayuda de la aplicación de Windows Form en C# del software Microsoft Visual Studio®.

Para terminar, con la ayuda de la aplicación HMI se ha observado la evolución del sistema y se ha procedido a la sintonización de las constantes del regulador PID.

5. DESARROLLO DEL PROYECTO

5.1. ELEMENTOS CONSTRUCTIVOS DEL PROYECTO

Para comenzar la elaboración del proyecto se ha dispuesto de los siguientes elementos:

- Taladradora Dremel® 300.
- Arduino Mega 2560.
- Sensor QRD 1113 incorporado en el taladro para medir la velocidad.

Para poder desarrollar el control se ha necesitado disponer de los siguientes elementos electrónicos:

- Optoacoplador FOD814 para el circuito de detección de cruce por cero de la onda de alimentación.
- Optoacoplador MOC3023 para el circuito de disparo del TRIAC.
- Transistor BC548 para el circuito de acondicionamiento del sensor QRD 113.
- TRIAC BTA 06 para variar la tensión del motor con el control de fase.

5.2. CIRCUITOS ELECTRÓNICOS Y CÁCULOS

Los circuitos necesarios para llevar a cabo este trabajo son tres, un circuito para la detección de cruce por cero de la onda de alimentación, un circuito para el acondicionamiento de la señal del sensor “encoder”, y un circuito actuador del sistema para variar el voltaje de alimentación del motor.

Es importante tener en cuenta que el circuito de detección de cruce por cero y el circuito de actuador hacen de interfaz entre la red eléctrica y el circuito de muy baja tensión del microcontrolador, por lo que es muy conveniente que se realice algún tipo de aislamiento. En este caso se ha optado por un aislamiento óptico mediante optoacopladores debido a su sencillez.

5.2.1. CIRCUITO DE DETECCIÓN CRUCE POR CERO

Como se ha mencionado antes, para poder analizar la señal de la red de alimentación sin correr peligro de estropear el Arduino es necesario aislarlo con un optoacoplador. Un optoacoplador es un elemento que incluye dos LEDS conectados en antiparalelo enfrentados con un fototransistor que conduce cuando alguno de los LEDS emite luz, por lo que es ideal para detectar pasos por cero de una onda sinusoidal como la de la red, sin tener que añadir más componentes adicionales que resistencias.

Para este circuito se ha escogido el optoacoplador FOD814 Según su datasheet [10] este componente posee las siguientes características representadas en la tabla 1.

Símbolo	**Parámetro**	**Valor**	**Uds.**
EMISOR			
$I_{F,max}$	Continuous Forward Current	50	mA
$V_{F,max}$	Forward Voltage	1,2-1,4	V
DETECTOR			
$I_{C,max}$	Continuous Collector Current	50	mA
$V_{CEO,sat}$	Collector-Emitter Voltage	70	V
CTR (I_F = 1mA)	Current Transfer Ratio	20-300	%

Tabla 1: Características del fototransistor FOD 814.

Como no se puede colocar el fototransistor directamente a la alimentación para no deteriorarlo, es necesario colocar en serie con dos resistencias (Rp y Rz) dimensionadas correctamente para su óptimo funcionamiento, tal y como se puede ver en la figura 9. Estas resistencias se dimensionan de la siguiente manera:

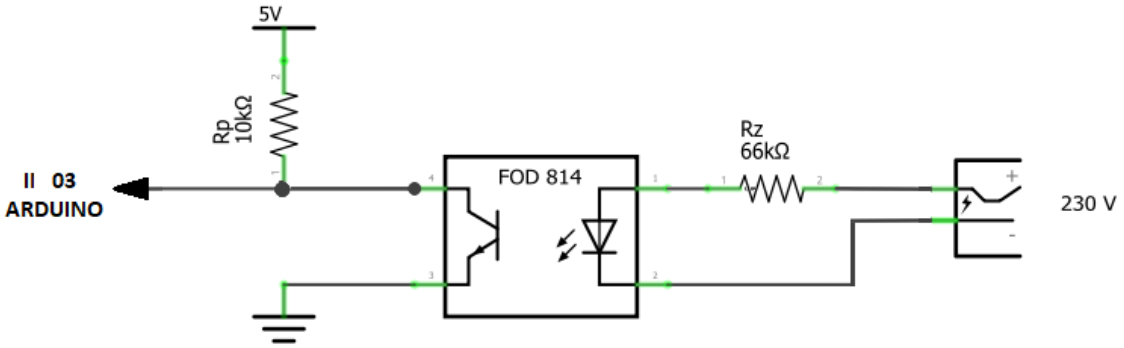


Figura 8: Circuito de detección de cruce por cero.

- Calculo de R_P:

  Limitando la corriente I_C a 0,5 mA:

$$R_P > \frac{V_{cc} - V_{CEO,sat}}{I_c} = \frac{5 - 0{,}2}{0{,}5mA} = 9\ k\Omega$$

$$R_P = 10\ k\Omega$$

- Calculo de R_Z:

Sabiendo que:

$f = 50\ Hz\ ;\ T = 20\ ms;\ V_{RMS} = 230\ V\ ;\ V_p = V_{RMS} \cdot \sqrt{2}\ \pm 6\%$

$$V_p = 230 \cdot \sqrt{2}\ = 325{,}3\ V \qquad V_p \pm 6\%$$

$$= \begin{cases} 344{,}8\ V \\ 305{,}8\ V \end{cases}$$

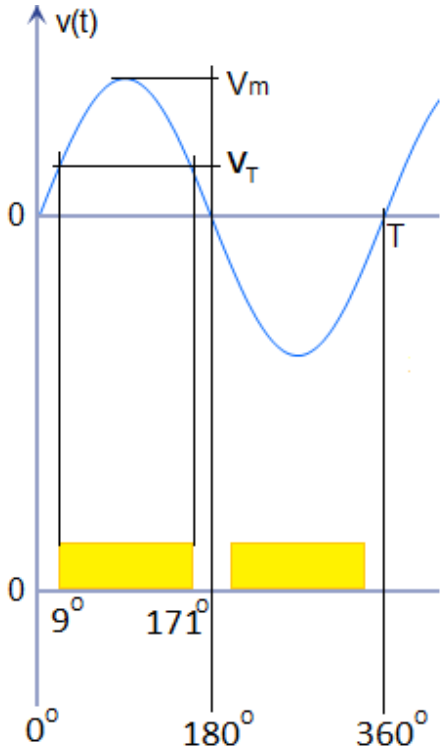


Figura 9:Diagrama de tensión y ángulos de disparo

$$V(t) = V_p \cdot sen(\omega t) = V_p \cdot sen(\alpha)$$

Y calculando que se active el LED entre el 5% y el 95 %, se puede calcular la tensión mínima a la cual estará sometido el fototransistor (figura 8):

$$\alpha_1 = 10ms \cdot 95\% = 9{,}5ms\ \rightarrow 171°$$

$$\alpha_2 = 5ms \cdot 5\% = 0{,}5ms\ \rightarrow 9°$$

$$V_{Tmin} = V_{p,min} \cdot sen(\alpha_1) = 305{,}8 \cdot sen(171°)$$

$$V_{Tmin} = 47{,}8\ V$$

$$V_{Tmin} = V_{p,min} \cdot sen(\alpha_2) = 305{,}8 \cdot sen(9°)$$

$$V_{Tmin} = 47{,}8\ V$$

Con esta tensión ya se puede calcular la resistencia R_Z:

$$R_z = \frac{V_{Tmin} - V_{Fmax}}{I_F} = \frac{47{,}8 - 1{,}4}{0{,}001} = 46400\ \Omega$$

$$I_{F,max} = \frac{V_{P,max} - V_{F,min}}{R_z} = \frac{344{,}8 - 1{,}2}{46400} = 7{,}4\ mA$$

$$P_{Rz,max} = I_{F,max} \cdot V_{z,max} = 7{,}4 \cdot 10^{-3} \cdot (344{,}8 - 1{,}2) = 2{,}54\ W$$

Esta potencia es demasiada para una sola resistencia, por ello en lugar de una sola resistencia se emplearán varias dispuestas como se puede ver en la figura 10.

$$R_a = \frac{R_z}{2} = 2310 \rightarrow 22\ k\Omega \text{ o } 25\ k\Omega$$

$$P_{R_a} = \frac{P_{Rz,max}}{n_{resistencias}} = \frac{2{,}54}{8} = 0{,}317\ W$$

$$V_{R_a} = \frac{V_{R_z}}{4} = \frac{(344{,}8 - 1{,}2)}{4} = 85{,}9\ V$$

Estas nuevas resistencias soportan tensiones y potencias necesarias.

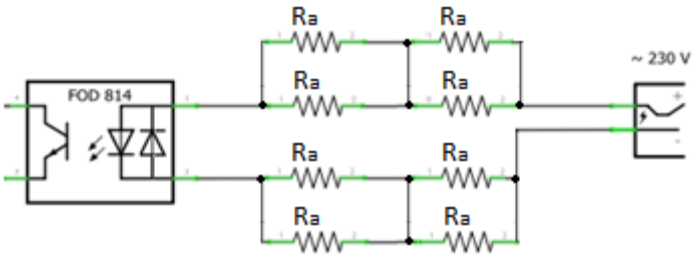


Figura 10: Sustitución de R_z. $R_z = 2 \cdot R_a$

5.2.2. CIRCUITO DEL SENSOR DE VELOCIDAD

El Sensor QRD113 empleado para la medición de la velocidad posee las siguientes características representadas en la tabla 2 según su datasheet [11].

Símbolo	Parámetro	Valor	Uds.
EMISOR			
$I_{F,max}$	Continuous Forward Current	50	mA
$V_{F,max}$ (I_F=20mA)	Forward Voltage	1,7	V
DETECTOR			
I_C (I_F=20mA)	Continuous Collector Current	0,3	mA
$V_{CEO,sat}$	Collector-Emitter Voltage	0,4	V
$V_{CEO,max}$	Max Collector-Emitter Voltage	30	V

Tabla 2: características eléctricas del sensor QRD113.

Como las variaciones de corriente del colector del sensor son pequeñas, se ha empleado un transistor para amplificar e invertir la señal, ya que el sensor ya está cableado en la Dremel con el negativo en común y de esa manera su transistor pone un 0V cuando detecta luz.

El transistor (Q1) elegido ha sido el BC548, y posee las siguientes características representadas en la tabla 3 según su datasheet [12].

Símbolo	Parámetro	Valor	Uds.
$Ic_{,max}$	Max Collector Current	100	mA
B (I_F=2mA)	Efficiency	110 - 800	
$V_{CEO,sat}$	Collector-Emitter Voltage	0,4	V
V_{BE}	Base -Emitter Voltage	0,7	V

Tabla 3: características eléctricas del sensor BC548.

Sabiendo las características de los componentes se pueden calcular los valores de las resistencias y condensadores del circuito de la figura 11.

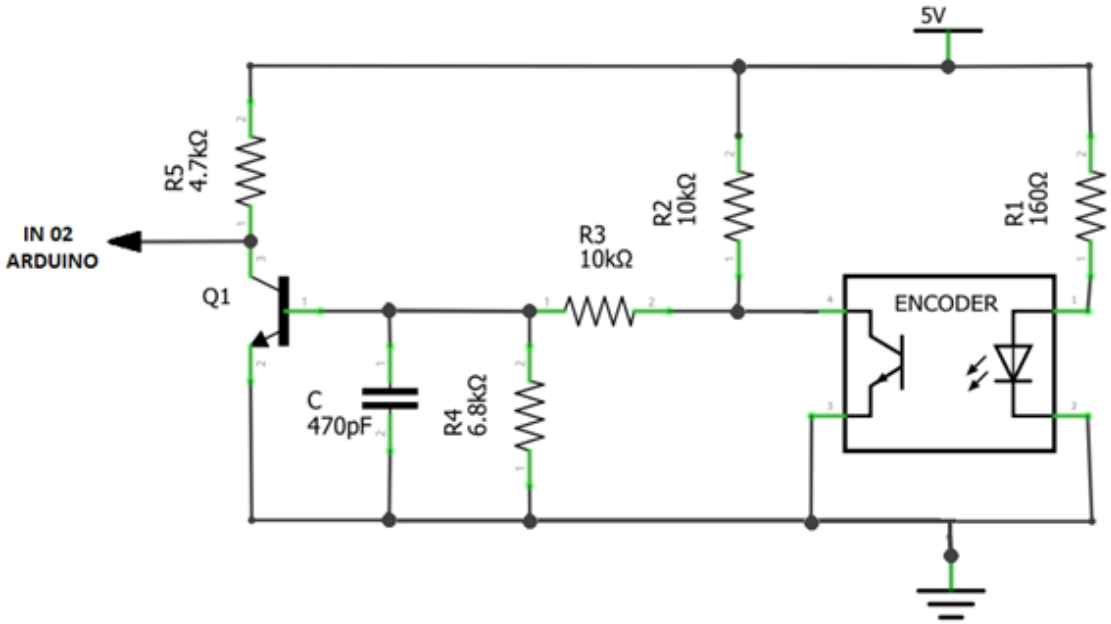


Figura 11: Circuito del sensor de velocidad.

$$R_1 = \frac{5 - V_F}{I_F} = \frac{5 - 1{,}2}{0{,}02} = 165\ \Omega \approx 160\ \Omega$$

Limitando la corriente I_{C2} a 1 mA y suponiendo una V_{CE2} = 0,2 V se puede calcular R_5 y suponiendo que el transistor del sensor este en estado "OFF":

$$R_5 = \frac{5 - V_{CE,Q2}}{I_C} = \frac{5 - 0{,}2}{0{,}001} = 4800\Omega \approx 4{,}7k\Omega \ \rightarrow \mathrm{I_{C2}} = \frac{5 - V_{CE,Q2}}{R_5} = \frac{5 - 0{,}2}{4700} = 1{,}02mA$$

$$I_{B2} = \frac{I_{c2}}{B} = \frac{0{,}00102}{110} = 9{,}3\mu A$$

Para asegurar la conducción de Q2 se toma: $\mathrm{I_{B2}} = 10 \cdot 9{,}3\mu A \cong 0{,}1\mathrm{mA}$

Si tenemos en cuenta que $I_4 \approx I_B$ entonces:

$$R_4 = \frac{V_{BE}}{I_4} = \frac{0{,}7}{0{,}0001} = 7k\Omega \ \approx 6800\Omega \ \rightarrow \mathrm{I_4} = \frac{0{,}7}{6800} = 0{,}103mA$$

$$R_2 + R_3 = \frac{(5 - V_{BE})}{I_3} = \frac{(5 - V_{BE})}{I_4 + I_B} = \frac{5 - 0{,}7}{0{,}0002} = 21500\Omega$$

Y ahora en el caso de que el transistor del sensor este en estado "ON":

$$R_2 = \frac{5 - V_{CE,sat}}{I_{C1}} = \frac{5 - 0{,}4}{0{,}0005} = 9200\Omega \approx 10\text{k}\Omega \rightarrow \text{I}_{\text{C1}} = \frac{5 - 0{,}4}{10000} = 0{,}46mA$$

$$R_3 = 21500 - R_2 = 21500 - 10000 \approx 10k\Omega$$

Para eliminar las señales de ruido de frecuencias mayores a la de los pulsos a medir se emplea un filtro paso bajo. Ahora solamente queda calcular el condensador del filtro paso bajo. Para ello hay que tener en cuenta la frecuencia a la que trabajará:

$$33000\ rpm \rightarrow 550\ rps(teniendo\ en\ cuenta\ solo\ un\ pulso\ por\ vuelta)$$

$$Frecuencia\ máxima\ de\ pulsos\ (f_m) = 550Hz$$

$$R_T = (R_2 + R_3)||R_4 = \frac{21500 \cdot 3800}{21500 + 6800} = 5160\ \Omega$$

Para que el condensador se pueda cargar y descargar más rápido que el menor periodo de pulsos (frecuencia máxima), hay que hacer que la constante de tiempo τ del circuito RC sea lo bastante pequeña.

$$5 \cdot \tau = 5 \cdot RC \ll \frac{1}{f_m} = 1{,}8ms$$

$$C \ll \frac{0{,}0018}{5 \cdot 5160} = 0{,}7nF \rightarrow 470pF\ (es\ mas\ facil\ de\ encontrar).$$

5.2.3. CIRCUITO DEL CONTROLADOR DE FASE

Se ha escogido el MOC3023 para esta aplicación por ser un optoacoplador muy común en este tipo de aplicaciones, para disparar triacs, y es económico. Además, este modelo no incluye un detector de paso por cero dentro, como otros, porque eso debe gestionarlo el Arduino.

El circuito de la figura 12 se ha sacado del datasheet [13] del Optoacoplador MOC3023, ya que el fabricante lo aconseja para un triac sensible como el que se empleará, añadiendo un filtro RC para proteger al triac del posible ruido producido por el motor.

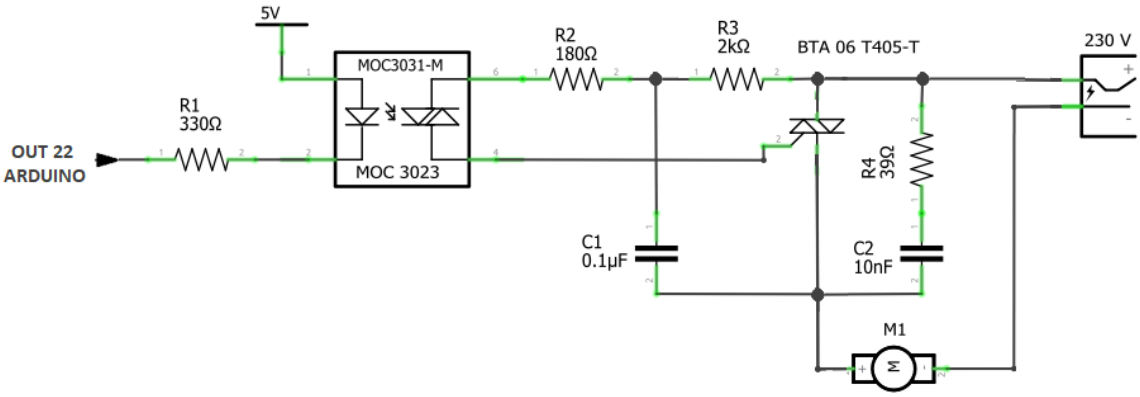


Figura 12: Circuito de disparo del TRIAC.

5.3. INTERFAZ DE MONITORIZACIÓN

Para la monitorización del sistema se ha optado por la elaboración de un entorno gráfico elaborado con la aplicación de Windows Form C# de Microsoft Visual Studio como se puede ver en la figura 13.

Esta interfaz contiene las siguientes funcionalidades:

1) Botón de marcha.
2) Botón de paro.
3) Botón de conexión.
4) Lista desplegable para la selección de los puertos COM disponibles.
5) Cuadro de texto para introducir la consigna de velocidad.
6) Botón para enviar la consigna de velocidad al Arduino.
7) Cuadros de texto para visualizar a cada momento la velocidad, error y señal de control.
8) Una gráfica en la que se visualiza la evolución de la velocidad del motor en rps en el eje de ordenadas y el tiempo en decima de segundos en el eje de abscisas. Esta gráfica se reinicia cada vez que se pulsa el botón de marcha
9) Cuadro de texto para la visualización de mensajes.

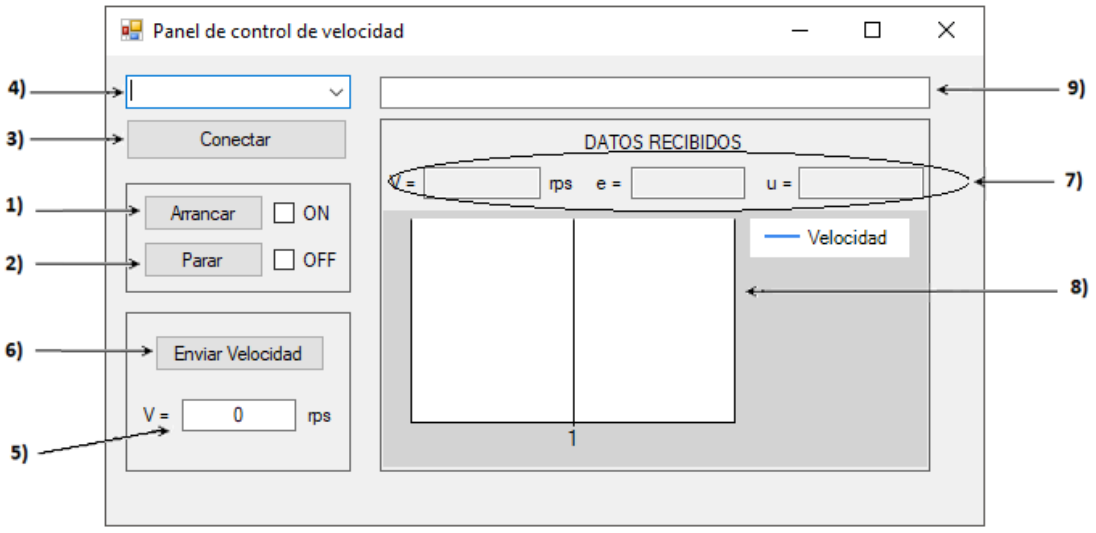


Figura 13: Interfaz HMI para la monitorización del sistema.

Para realizar la interfaz primeramente he creado una ventana y he posicionado los distintos elementos, como botones, cuadros de texto, desplegables, etc. Una vez colocados todos lo elementos le he configurado las propiedades, nombre, texto, apariencia, etc. Mas tarde he introducido y configurado un puerto serie y un "timer".

Con todos los elementos introducidos y parametrizados se les ha asignado eventos que desencadenan acciones. Por ejemplo, cuado se pulsa el botón "Enviar velocidad" se envía por el puerto serie el texto introducido en el cuadro de texto de la velocidad.

En el caso de la gráfica, se ha configurado una serie llamada velocidad de manera que el gráfico sea de tipo Spline, y se ha programado de forma que, si el check box "on" esta chequeado se introduce en la serie el valor de la velocidad leída del Arduino en el eje y, y se incrementa en uno el eje x. En el caso de que se pulse el botón paro la gráfica dejará de introducir valores y al volver a pulsar el botón marcha se reseteará.

Para la comunicación entre la aplicación HMI y el Arduino se ha optado por una comunicación serie en la que se envía y recibe la información mediante comandos uno tras otro de forma separada y bajo demanda. Por ejemplo, si se recibe una "M" quiere decir que el sistema esta en marcha y en el caso de enviar una "V" seguida de un número se esta enviando la velocidad de referencia. El código de esta aplicación se puede ver en el ANEXO 1.

5.4. IMPLEMENTACIÓN EN ARDUINO

En este apartado se verá la implementación del código necesario para llevar a cabo el control del motor, desde la medición de su velocidad, hasta el accionamiento de este. El código completo se puede consultar en el ANEXO 2.

5.4.1. MEDICIÓN DE LA VELOCIDAD

La medición de la velocidad se ha realizado de manera que cuando el encoder manda un flanco de bajada a la entrada digital 2 del Arduino MEGA, se incrementa en uno el valor de un contador (PASOS). Cuando el valor del contador "PASOS" es igual a cero se guarda el valor del tiempo en una variable (tiempo1) y cuando el valor del contador "PASOS" es igual a un numero de vueltas, en este caso 10, se guarda el valor del tiempo en otra variable (tiempo2) y se calcula la velocidad, en revoluciones por segundos, de manera que esta es igual al número de vueltas dividido entre el incremento del tiempo (tiempo2 – tiempo1). Una vez calculada la velocidad se resetea el contador "PASOS".

```
//** INTERRUPCION DE LECTURA DEL ENCODER **//
void encoder() {
  if (PASOS == 0)
    tiempo1 = millis();

  PASOS ++;
  if (PASOS >= vuelta) {
    tiempo2 = millis();
    tiempoVuelta = (tiempo2 - tiempo1);
    velocidad = (float)(vuelta * 1000 / tiempoVuelta); // rps
    PASOS = 0;
  }
```

Código 2: Función de lectura del encoder y cálculo de velocidad.

5.4.2. REGULADOR PID

El regulador PID implementado en este caso es un regulador PID clásico, como el que se menciona en el apartado 2.1. Además, se han limitado las señales de control y escalado para que la máxima y mínima señal de control correspondan con un retardo en el disparo del Triac de 0ms y 10ms respectivamente.

$$u_{(t)} = K_p \cdot e_{(t)} + K_i \cdot T_S \sum_{k=0}^{t} e_k + K_d \frac{e_{(t)} - e_{(t-1)}}{T_S}$$

```
void controlador() {

  tiempo3 = millis();
  dT = (double)(tiempo3 - tiempo4);

    if (consigna > vmax) {
      consigna = vmax;
    }
    if (consigna < 0){
    consigna = 0;
    }

 //* P *//
    error = consigna - velocidad;
    P = kp * error;

 //* I *//
    Ipar = ki * (error+errAnterior)*dT/1000;
    if ((I+Ipar < umax)&&(I+Ipar> umin)){
      I += Ipar;
      }
    else
      I = I;

//* D *//
    errDif = (error - errAnterior)*1000 / dT;
    D = (float)(kd * errDif*1000 / dT );

 //* PID *//
    u = P + I + D; // señal del controlador sin prealimentación

 // Limitacón de la señal de control //
  if (u > umax) {
    u = umax;
  }
  else if (u < umin) {
    u = umin;
  }
  else
    u = u;

  c = (300-(300*u / umax) ); // escalado de señal a microsegundos para el
disparo del triac

if ((c<=1))
  c=1;
if (c>300)
  c=300;
```

```
OCR1A = c;
 //* Actualización de variables para la proxima iteración
  errAnterior = error;
  tiempo4 = tiempo3;
  }
```

Código 3: Regulador PID implementado.

5.4.3. DETECCIÓN DE PASO POR CERO DE LA RED ELÉCTRICA Y DISPARO DEL TRIAC

La detección de paso por cero se realiza mediante una interrupción, cuando se detecta por la entrada digital 3 un flanco de bajada.

Para realizar el disparo con el retardo de tiempo previamente calculado por el regulador se ha empleado el temporizador “Timer 1” del Arduino. Con este método se consigue mayor precisión que empleando librerías estándar de temporización como la MSTimer2, ya que estas librerías suelen tener una resolución de milisegundos frente a la resolución de microsegundos que se puede alcanzar configurando el temporizador a bajo nivel.

Entonces una vez ejecutada la interrupción de detección del paso por cero, se activa el temporizador y cuando este llega a la cuenta calculada se activa, la salida 22 del Arduino, que es la salida correspondiente al terminal “puerta” del Triac. Después de un breve tiempo se desactiva el triac y el temporizador hasta que se vuelve a activar la interrupción de detección de paso por cero.

```
//**** Interrupción de detección de pasovelocidada por cero ****//
void cero() {

  TCCR1B = 0x04; // inicia el temporizador con divide por entrada de 256
  TCNT1 = 0; // reiniciar el temporizador - contar desde cero
}


//***************** Disparo del Triac  **********************//

ISR (TIMER1_COMPA_vect) {// comparador coincide con
  digitalWrite (TRIAC, HIGH); // establece la puerta TRIAC en
  TCNT1 = 65536-8; // ancho de impulso de disparo 16 bits
}

ISR(TIMER1_OVF_vect){ //timer1 overflow
  digitalWrite(TRIAC,LOW); // apaga la puerta TRIAC
  TCCR1B = 0x00;          // deshabilita el cronómetro
}
```

Código 4: Detección de paso por cero de la red y disparo del Triac.

5.4.4. COMUNICACIÓN

Para la comunicación ente Arduino y el panel de control HMI se emplea una comunicación basada en cadenas simples de caracteres ASCII. Por un lado, el Arduino envía cadenas con su estado, velocidad, error y señal de control, y por otro lado espera comandos de marcha, paro y la velocidad de referencia del sistema, desde el PC a los que deberá atender.

Esta información se envía desde Arduino al PC de la siguiente manera:

- El estado de marcha se indica enviado el carácter ASCII “M”.
- El estado de paro se indica enviado el carácter ASCII “P”.
- La velocidad se envía con el carácter ASCII “V” seguida de la velocidad del motor en rps.
- El error se envía con el carácter ASCII “E” seguida del error del sistema en rps.
- La señal de control se envía con el carácter ASCII “C” seguida de la señal de control.

Los comandos recibidos por el Arduino desde el PC son de la siguiente manera:

- Las ordenes de marcha y paro se realizan recibiendo el carácter ASCII “M” y “P” respectivamente.
- La consigna se recibe con el carácter ASCII “V” seguida de la velocidad deseada en rps.

```
void monitorizacion() {

if (millis()-tiempo_mon >= PERIODO_MON ) {
    Serial.print("P");
    Serial.println(paro);
    Serial.print("M");
    Serial.println(marcha);
    Serial.print("V");
    Serial.println(velocidad);
    Serial.print("C");
    Serial.println(u);
    Serial.print("E");
    Serial.println(error);
    Serial.print("Delay Triac  ");
    Serial.println(c);
    tiempo_mon=millis();
  }
  //********* RECIBIR DATOS DE PC *************

  while (Serial.available() > 0) {
    datoSerial = Serial.read();
    if (datoSerial == 'M')
      marcha = true;

    if (datoSerial == 'P')
      STOP();

    if (datoSerial == 'V') {
      delay(10);
      consigna = 0;
      datoSerial = Serial.read();
      while (datoSerial != '/') {
        consigna = consigna * 10 + (datoSerial - '0');
        datoSerial = Serial.read();
      }
    }
  }
}
```

Código 5: Comunicación del Arduino.

5.5. CONFIGURACIÓN Y SINTONIZACIÓN DEL PID

Como se ha mencionado en el punto 2.1. de este trabajo, es necesario realizar ciertas configuraciones y sintonizar o determinar las constantes proporcional, integral y derivativa del PID, para poder implementarlo.

Debido a que el motor siempre gira en el mismo sentido, no tendría sentido que la señal de control fuese negativa. Por ello se ha limitado la señal de control (u) en valores de entre 1000 y 0.

Posteriormente se ha escalado la señal de control entre 1 y 300 que corresponden a los pulsos de la cuenta del "Timer 1", que, a su vez, corresponden al retardo de disparo de entre 0 y 10ms.

$$c = 300 - \left(300 * \frac{u}{umax} umax\right) = pulsos\ de\ cuenta\ del\ Timer\ 1$$

Y, para terminar, se han condicionado a que los valores mínimo y máximo de la escala de manera que "c" no exceda de 300 y no disminuya de 1. El valor de "c" no puede ser menor que 1 debido a que es el mínimo valor para que el timer cuente al menos una vez de 1 a 0, y no tiene sentido un tiempo menor a 0s.

De esta manera 1 correspondería a nada de retardo de disparo y 300 correspondería a 10ms de retardo.

Para realizar la correcta sintonización del motor, es necesario identificar su modelo matemático, pero modelar el motor AC empleado, que no es precisamente de precisión, es mas complejo que modelar un motor DC y requeriría demasiado tiempo, por ello se ha decidido que esto quede como un trabajo futuro.

Cómo la sintonización no se puede hacer de forma matemática, se deberá realizar de forma iterativa. Para ello se siguen los siguientes pasos:

Primeramente, se asigna un valor de 0 a las constantes K_P, K_I y K_d. Después, se incrementa la constante proporcional hasta obtener una respuesta cercana a la referencia establecida, aunque esto no elimine el error en régimen estacionario. En este punto es posible que aparezca algún sobreimpulso u oscilaciones.

Para corregir el error en estado estacionario se incrementa el valor de la constante integral hasta eliminar el offset en el tiempo deseado. Es posible que aumenten el sobre impulso y las oscilaciones.

Para reducir el sobre impulso y las oscilaciones que puedan haber surgido, se ajusta la constante derivativa.

Por último, se termina de ajustar los valores de las constantes para afinar la respuesta a la deseada.

En el siguiente apartado se mostrarán los estos pasos llevados a la práctica.

6. PRUEBAS

Para la realización de las pruebas se han montado los circuitos en placas de pruebas como se puede ver en la figura 14.

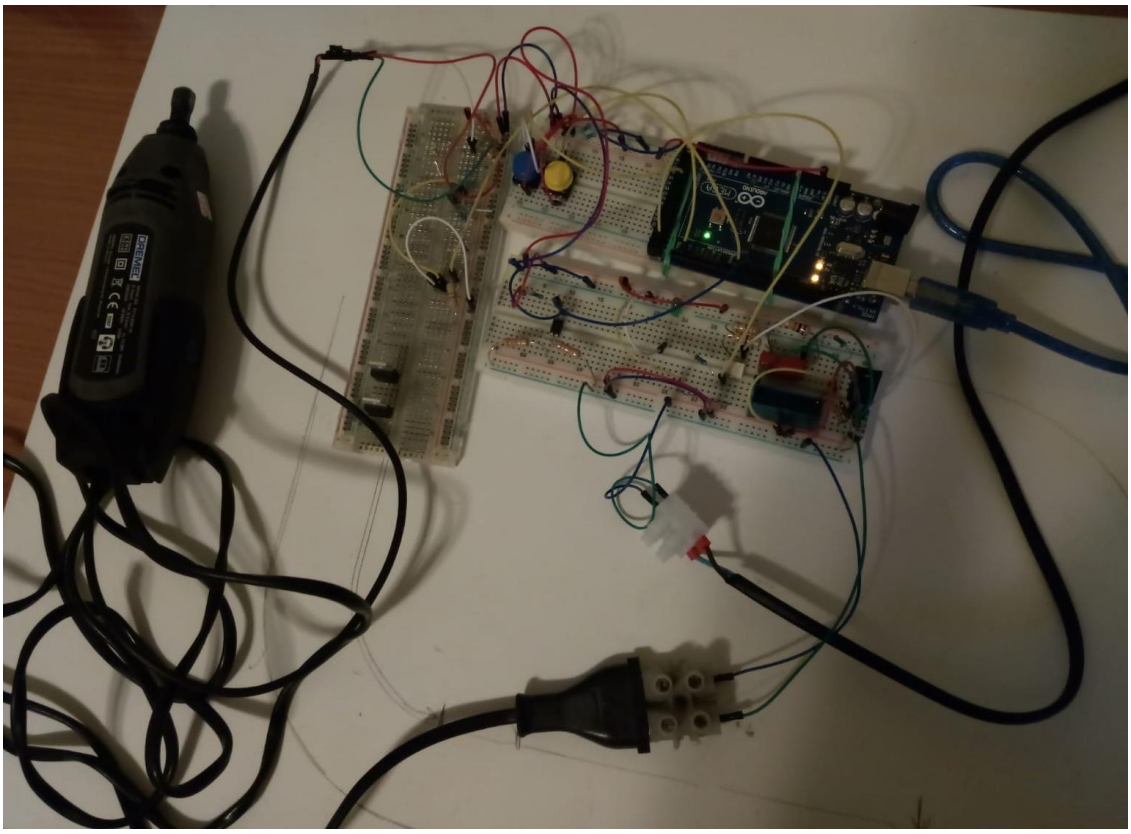


Figura 14: Montaje del proyecto.

Una vez montado se ha procedido a la observación del sistema, y la calibración de las constantes del PID. Estos ensayos se han realizado sin carga en el motor, y las constantes se han modificando en el código de Arduino y se ha monitorizado desde el entorno HMI creado previamente.

Como se ha explicado en el apartado 5.5. para la sintonización del PID, primero se han establecido las constantes a 0 y posteriormente se ha aumentado la constante kp hasta kp=3 para que el sistema se acerca a la consigna establecida (figura 15).

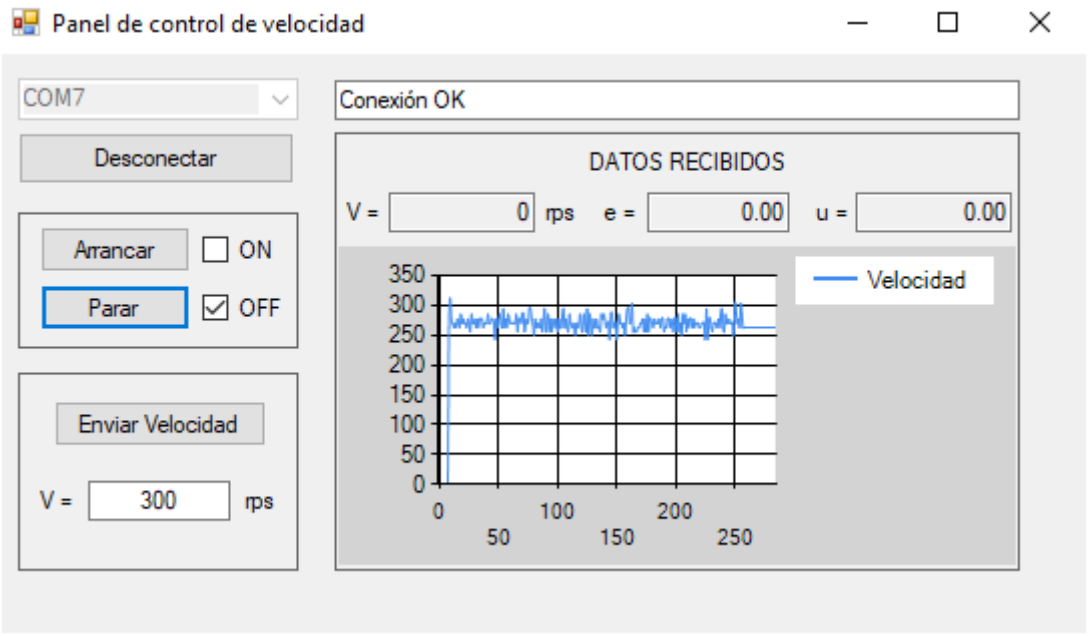


Figura 15: respuesta del sistema para un escalón de 0 a 300rps con unos valores de kp=3, ki=kd=0.

Posteriormente para eliminar el offset se ha aumentado la constante se ha asignado a la constante integral un valor de ki=1 obteniendo la respuesta de la figura 16.

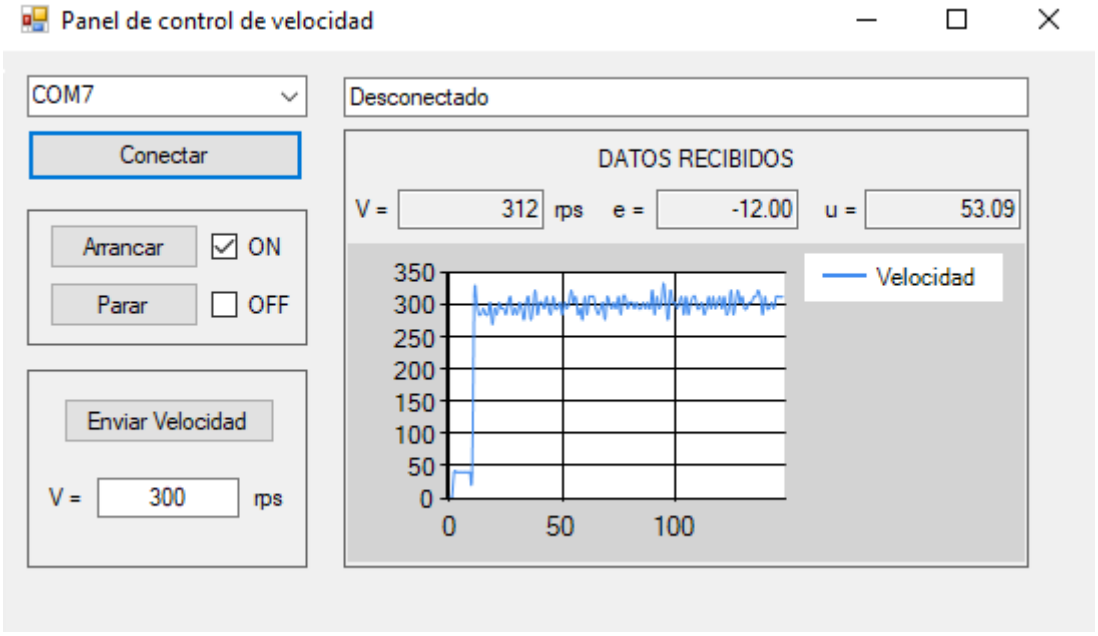


Figura 16: respuesta del sistema para un escalón de 0 a 300rps con unos valores de kp=3, ki=1 y kd=0.

Ahora se ha probado aumentado la constante derivativa a diversos valores pero, en ningún caso, se ha obtenido una mejora en la oscilación del sistema. Como se puede ver en la figura 17 el aumento de la constante derivativa hace que el sistema sea más oscilante, y

por ello se ha decidido eliminar esta constante o lo que es lo mismo kd=0. El eliminar la constante derivativa quiere decir que el control finalmente será un regulador PI.

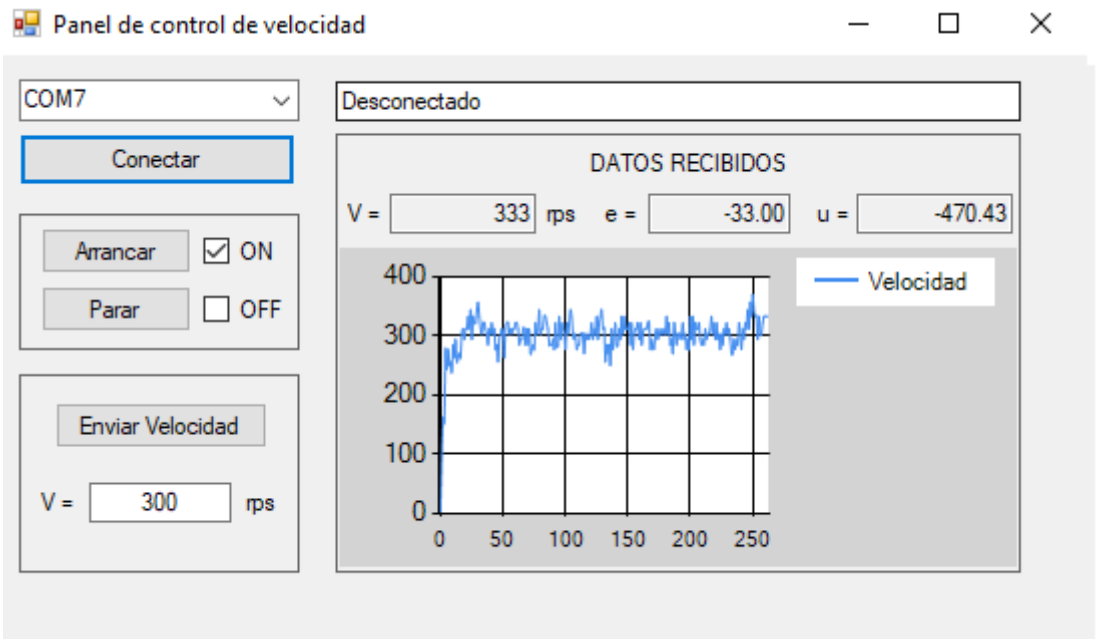


Figura 17: respuesta del sistema para un escalón de 0 a 300rps con unos valores de kp=3, ki=1 y kd=0,1.

En la figura 18 se puede observar el comportamiento del sistema sin carga ante varios cambios de la consigna. De esta respuesta se puede deducir que el motor se establece alrededor de las referencias bastante rápido y también se puede ver que el motor oscila en menor medida cuando la consigna es más cercana a su velocidad nominal (33000rpm o 550rps). Debido a esto último he establecido el régimen de funcionamiento entre 550rps y 200rps (12000rpm).

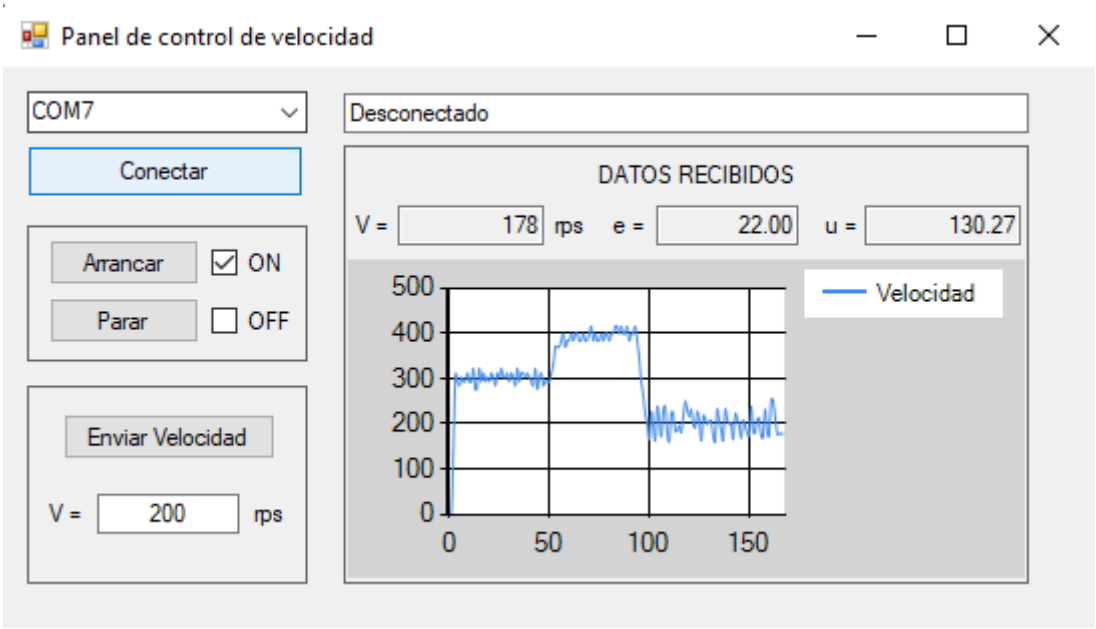


Figura 18: respuesta del sistema para un escalón de 0 a 300rps y posteriormente un cambio de referencia a 400rps y 200rps (kp=3 y ki=1).

- Ensayo en carga:

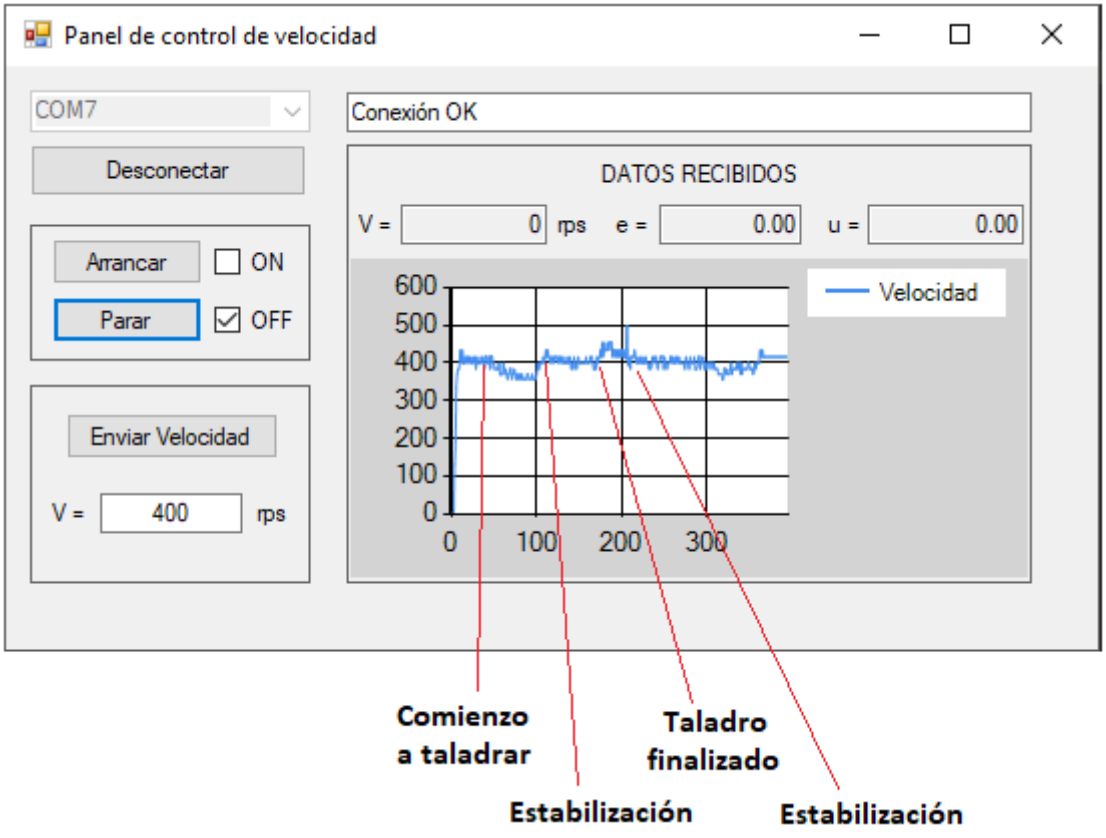


Figura 19: Comportamiento del sistema taladrando una figura de madera.

El ensayo en carga se ha realizado taladrando una figura de madera ya que este es el uso para el que estaba diseñado el motor. En la figura 19 se puede observar como el motor se frena al comenzar a taladrar ya que aumenta el par resistente, posteriormente se produce una estabilización entorno a la consigna y al finalizar el taladrado, como desaparece el par resistente aumenta drásticamente la velocidad produciéndose una sobreoscilación, que posteriormente la acción de control corrige.

Aparentemente el control funciona bastante bien, aunque no se ha realizado el sintonizado de manera matemática.

7. PESUPUESTO

El siguiente presupuesto de la tabla 4 se ha realizado con los precios de venta sin IVA y sin tener en cuenta los gastos de envío de los componentes.

MATERIAL	PRODUCTO	FABRICANTE	PRECIO UD	UDs	PRECIO
Fototransistor	QRD1113	ON SEMICONDUCTOR®	2,44 €	1	2,44 €
Optoacoplador	FOD814	ON SEMICONDUCTOR®	0,39 €	1	0,39 €
Optoacoplador	MOC3023	ON SEMICONDUCTOR®	0,47 €	1	0,47 €
Transistor	BC548	ON SEMICONDUCTOR®	0,15 €	1	0,15 €
Triac	BTA06	ST Microelectronics®	1,14 €	1	1,14 €
Resistencia	4,7kohms 250V 600mW	Multicomp®	0,03 €	1	0,03 €
Resistencia	10kohms 250V 600mW	Multicomp®	0,03 €	3	0,09 €
Resistencia	39ohms 350V 600mW	VISHAY®	0,07 €	1	0,07 €
Resistencia	330ohms 500V 2W	Multicomp®	0,14 €	1	0,14 €
Resistencia	6,8kohms 500V 2W	VISHAY®	0,27 €	1	0,27 €
Resistencia	160ohms 250V 600mW	Multicomp®	0,03 €	1	0,03 €
Resistencia	180ohms 350V 1W	Multicomp®	0,14 €	1	0,14 €
Resistencia	2kohms 350V 1W	Multicomp®	0,13 €	1	0,13 €
Resistencia	25kohms 350V 3W	Multicomp®	0,40 €	1	0,40 €
Condensador	0,1uF 500V	KEMET®	1,94 €	1	1,94 €
Condensador	10nF 500V	AVX®	1,94 €	1	1,94 €
Condensador	470pF 50V	VISHAY®	0,26 €	1	0,26 €
Arduino	MEGA 2560	Arduino	8,95 €	1	8,95 €
				TOTAL	**18,98 €**

Tabla 4: Presupuesto basándose en [14] a fecha de 20/06/2018.

8. CONCLUSIONES

Aunque se pueden encontrar bastantes proyectos de control de fase de un motor AC mediante Arduino u otros controladores, todos los que se han encontrado son en bucle abierto, sin que realmente tengan controlador ni se tenga en cuenta la velocidad real del motor.

A pesar de ello, en este trabajo se comprueba que se puede realizar un control de fase de un motor AC monofásico por medio de un PID implementado en Arduino, y se puede ver que ha dado buenos resultados, a pesar de las oscilaciones entorno a la consigna. Estas oscilaciones pueden ser consecuencia de la poca resolución del sensor de medida de la velocidad y de que este es un motor creado para poseer una dinámica y velocidad muy rápida, lo que conlleva a no poseer mucha precisión de velocidad a baja velocidad, de hecho, por debajo de 200rps (12000rpm) el motor no funciona correctamente con este sistema de control.

Cabe destacar que en este caso solo se ha requerido de un PI, en lugar de un PID, esto es debido a que en este caso el emplear la parte derivativa se incrementaban las oscilaciones en el control del motor.

Como se puede ver en la tabla 4 del presupuesto, este es un circuito de bajo coste, lo cual hace que para, sobre todo, aplicaciones de control de dinámica lenta sea un control muy asequible.

Para posibles trabajos futuros se podría hacer el modelado matemático del motor y el conjunto motor más Triac, que permitiría simular el sistema y sintonizar el PID de forma precisa mediante el método de cancelación de polos, o incluso hacer el control por medio del control en espacio de estado. También se podría añadir una pantalla LCD al Arduino para no tener la necesidad de tener un PC conectado al sistema, y también se podría integrar un encoder de mayor resolución y controlar la posición del motor.

9. REFERENCIAS

[1] O. Reinoso, Control de sistemas discretos. 2004.

[2] J. Fraile Mora, Máquinas Eléctricas. .

[3] N. Mohan, T. M. Undeland, y W. P. Robbins, Power Electronics. Converters, Applications, and Design, vol. 4. 2007.

[4] Activity 6 Part (b): PI Control of a DC Motor. http://ctms.engin.umich.edu/CTMS/index.php?aux=Activities_DCmotorB

[5] Control Speed of DC FAN Using Arduino PID Library. https://arduino.stackexchange.com/questions/9731/control-speed-of-dc-fan-using-arduino-pid-library

[6] Arduino PID Library. https://playground.arduino.cc/Code/PIDLibrary

[7] Zero-Crossing Detectors Circuits and Applications by Lewis Loflin. http://www.bristolwatch.com/ele2/zero_crossing.htm

[8] Arduino Hardware Interrupts Control AC Power by Lewis Loflin. http://www.bristolwatch.com/arduino/arduino_power_control.htm

[9] AC Phase Control. https://playground.arduino.cc/Main/ACPhaseControl

[10] Fairchild Semiconductors, «FOD814 Series, FOD817 Series 4-Pin DIP Phototransistor Optocouplers Features».

[11] Fairchild Semiconductors, «QRD1113-QRD1114 Reflective Object Sensor». .

[12] Fairchild Semiconductors, «Transistor Bc548».

[13] Fairchild Semiconductors, «6-Pin DIP Random-Phase Optoisolators Triac Driver Output MOC3010», Test, pp. 1-7, 1995.

[14] Farnell Website. http://es.farnell.com/

ANEXOS

ANEXO 1: Código de programación de la aplicación HMI

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using System.Windows.Forms;

namespace Interfaz
{
    public partial class Principal : Form
    {
        bool enviarVel = false;
        bool Para = false;
        bool Arranca = false;
        int Vel = 0;
        double rt = 0;

        public Principal()
        {
            InitializeComponent();
        }

        private void btnCon_Click(object sender, EventArgs e)
        {
            if (!PuertoSerie.IsOpen)
            {
                try
                {
                    PuertoSerie.Open();
                    ListaCOM.Enabled = false;
                    btnCon.Text = "Desconectar";
                    txtEstado.Text = "Conexión OK";
                    timer1.Enabled = true;
                }
                catch (System.Exception ex)
                {
                    txtEstado.Text = ex.Message;
                }
            }
            else
            {
                PuertoSerie.Close();
                ListaCOM.Enabled = true;
                btnCon.Text = "Conectar";
                txtEstado.Text = "Desconectado";
                timer1.Enabled = false;
            }
        }

        // Obtención de los Puertos Com disponibles //
        private void ListaCOM_Click(object sender, EventArgs e)
        {
            ListaCOM.Items.Clear();
```

```
            btnCon.Enabled = false;
            foreach (String coms in System.IO.Ports.SerialPort.GetPortNames())
            {
                ListaCOM.Items.Add(coms);
            }
        }

        // Asignacion del Puerto Com  //
        private void ListaCOM_SelectedIndexChanged(object sender, EventArgs e)
        {
            btnCon.Enabled = true;
            if (!PuertoSerie.IsOpen)
            {
                PuertoSerie.PortName = ListaCOM.SelectedItem.ToString();
            }
        }

        private void Principal_FormClosing(object sender, FormClosingEventArgs e)
        {
            PuertoSerie.Close();
        }


         private void timer1_Tick_1(object sender, EventArgs e)
        {
        String aux, lectura;
            //enviar Velocidad de referencia (consigna)
            if (enviarVel)
            {
                PuertoSerie.Write("V");
                PuertoSerie.Write(txtVel.Text);
                PuertoSerie.Write("/");       //finalizar trama de numeros
                enviarVel = false;
            }
            if (Arranca)
            {
                PuertoSerie.Write("M");
                Arranca = false;
            }
            if (Para)
            {
                PuertoSerie.Write("P");
                Para = false;
            }

            //recibir información del Arduino

            while (PuertoSerie.BytesToRead > 2)
            {
                //txtEstado.Text = "";
                try
                {
                    lectura = PuertoSerie.ReadLine();
                    //txtEstado.Text = lectura;
                    aux = lectura.Substring(1);
                    //txtEstado.Text = aux;
                    switch (lectura.First())
                    {
                        case 'M':
                            if (aux.CompareTo("1") == 1)
                                checkBox_ON.Checked = true;
                            else
```

```
                            checkBox_ON.Checked = false;
                        break;
                    case 'P':
                        if (aux.CompareTo("1") == 1)
                            checkBox_OFF.Checked = true;
                        else
                            checkBox_OFF.Checked = false;
                        break;

                        ////////////////////////////
                    case 'C':
                        txtControl.Text = aux;
                        break;
                    case 'E':
                        txtError.Text = aux;
                        break;
                    case 'V':
                        txtEncoder.Text = aux;
                        Vel = Int32.Parse(aux);
                        break;
                }
            }
            catch (System.Exception ex)
            {
                txtEstado.Text = ex.Message;
            }

        }
        if (checkBox_ON.Checked == true)
        {
            Historial.Series[0].Points.AddXY(rt, Vel);
            rt = rt + 1;
        }
    }

    private void btnEnviar_Click(object sender, EventArgs e)
    {
        enviarVel = true;
    }

    private void btnON_Click(object sender, EventArgs e)
    {
        Arranca = true;
        Historial.Series[0].Points.Clear();
    }

    private void btnOFF_Click(object sender, EventArgs e)
    {
        Para = true;
    }
}
}
```

ANEXO 2: Código de programación del Arduino

```
#include <interrupt.h>
#include <io.h>

//*************** ENTRADAS *********************************//
byte ENC = 2 ;
byte CERO = 3 ;
byte MARCHA = 44 ;
byte PARO = 19 ;

//*************** SALIDAS **********************************//
byte TRIAC = 22; //

// ****************** VARIABLES PID ***************************** //
int consigna = 0;  // rps
float kp = 3.0;            // Constante kP
float ki = 1.0;            // Constante ki
float kd = 0.0;            // Constante kd
float error = 0;          // Error (consigna-lectura)
float errAcum = 0;       // Acumulacion de errores
float errAnterior = 0;   // Error anterior
float errDif = 0;        // Diferencia de error actual y error anterior
float umax = 1000;       // Máxima señal de control
float umin = -1000;      // Minima señal de control
float dT = 0;         // Intervalo de tiempo de tiempo entre calculo del control
seg
float P = 0;             // Parte P del PID
float I = 0;             // Parte I del PID
float Ipar = 0;         // Integral parcial
float D = 0;             // Parte D del PID
float u = 0;             // Señal de control (P+I+D)
long c = 300;            // Señal de control escalada de 0 a 10ms en milisegundos

//************* VARIABLES ENCODER ***************************//
int PASOS = 0;          // Pasos detectados del encoder del motor
int vuelta = 10;        // Pasos por vuelta del encoder del motor
int tiempo1 = 0;        // Tiempo para calcular la velocidad
int tiempo2 = 0;        // Tiempo para calcular la velocidad
int tiempoVuelta = 0;   // Tiempo que tarda en dar una vuelta el encoder
int velocidad = 0;      // Velocidad calculada

//*********** VARIABLES AUXILIARES **********************//
int vmax = 33000 / 60;  //  Velocidad máxima en rpm/60 = rps
bool paro = true;      // Estado de parada
bool marcha = false;    // Estado de marcha
byte datoSerial = 0;    // Variable donde se guardan los datos recibidos
long tiempo3 = 0;       // Momento de Tiempo en el que se ejecuta el controlador
long tiempo4 = 0;       // Tiempo anterior al calculo del PID
#define PERIODO_MON 100 //ms
unsigned long tiempo_mon=0L;

//***************INICIALIZACION PROGRAMAS *****************//

void monitorizacion();
```

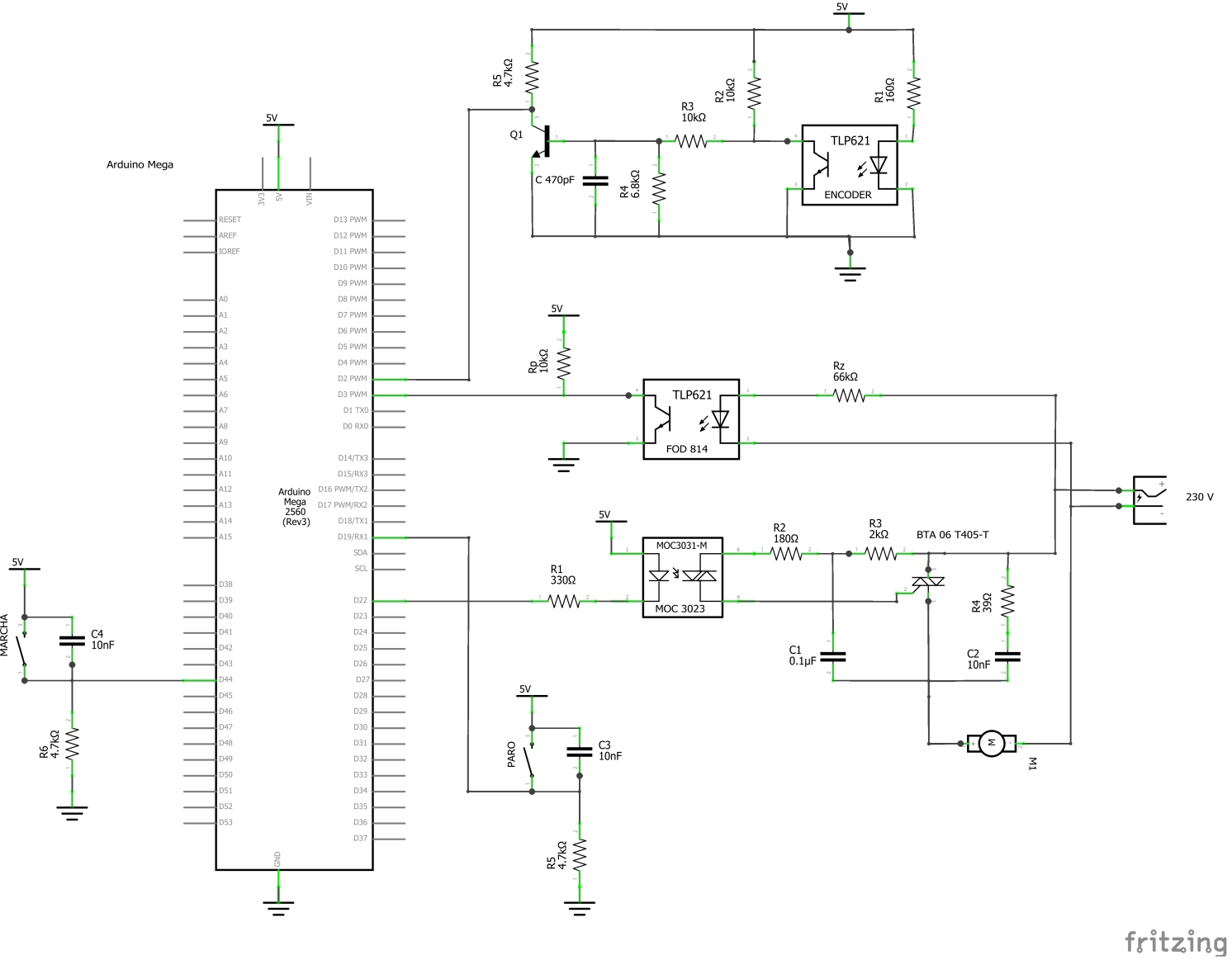
```
void controlador();
void encoder();
void cero();
void STOP();

//************* SETUP ************************//
void setup() {
  pinMode(ENC, INPUT);  //INT0    ENC
  pinMode(CERO, INPUT);  //INT1   PASO POR CERO
  pinMode(PARO, INPUT);  //INT2   PARADA
  pinMode(TRIAC, OUTPUT);
  digitalWrite(TRIAC, LOW); //Inicializacion del disparo del triac (off)

  // configuración Timer1
  // (ver la hoja de datos ATMEGA )
  OCR1A = 100; // inicializa el comparador
  TIMSK1 = 0x03; // habilitar el comparador A y las interrupciones de
desbordamiento
  TCCR1A = 0x00; // registros de control del temporizador establecidos para
funcionamiento normal
  TCCR1B = 0x00; // temporizador desactivado

  Serial.begin(9600);
  Serial.println("Encendido");

  //Interrupciones
  attachInterrupt(0, encoder, FALLING); // pin2 ENC
  attachInterrupt(1, cero, FALLING); // pin 3 CERO
  attachInterrupt(4, STOP, RISING); // pin 19 PARO
}

//***************** VOID LOOP ***************************//
void loop() {

  monitorizacion();

  if (paro == true) {
    //noInterrupts ();
    detachInterrupt(1);
    error = 0;
    errAcum = 0;
    errDif = 0;
    velocidad = 0;
    c = 300;
    consigna = 0;
    digitalWrite(TRIAC,LOW);
    delay(3000);
    interrupts();
  }

  if (digitalRead(MARCHA) || marcha) {
    //   Serial.println("marcha");
    paro = false;
    marcha = true;
  }

  if (!paro && marcha) {
    attachInterrupt(1, cero, FALLING); //
    controlador();
```

```
  }
}

//*********** MONITORIZACION DEL SISTEMA *****************//
void monitorizacion() {


if (millis()-tiempo_mon >= PERIODO_MON ) {
    Serial.print("P");
    Serial.println(paro);
    Serial.print("M");
    Serial.println(marcha);
    Serial.print("V");
    Serial.println(velocidad);
    Serial.print("C");
    Serial.println(u);
    Serial.print("E");
    Serial.println(error);
    Serial.print("Delay Triac  ");
    Serial.println(c);
    tiempo_mon=millis();
  }

  //********* RECIBIR DATOS DE PC *************

  while (Serial.available() > 0) {
    datoSerial = Serial.read();
    if (datoSerial == 'M')
      marcha = true;

    if (datoSerial == 'P')
      STOP();

    if (datoSerial == 'V') {
      delay(10);
      consigna = 0;
      datoSerial = Serial.read();
      while (datoSerial != '/') {
        consigna = consigna * 10 + (datoSerial - '0');
        datoSerial = Serial.read();
      }
    }
  }
 // delay(1000);
}

// ***************** PID *****************//
void controlador() {

  tiempo3 = millis();
  dT = (double)(tiempo3 - tiempo4);

    if (consigna > vmax) {
      consigna = vmax;
    }
    if (consigna < 0){
    consigna = 0;
```

```
    }

 //* P *//
    error = consigna - velocidad;
    P = kp * error;

 //* I *//
    Ipar = ki * (error+errAnterior)*dT/1000;
    if ((I+Ipar < umax)&&(I+Ipar> umin)){

      I += Ipar;
      }
    else
      I = I;

//* D *//
    errDif = (error - errAnterior)*1000 / dT;
    D = (float)(kd * errDif*1000 / dT );

 //* PID *//
    u = P + I + D; // señal del controlador sin prealimentación

 // Limitacón de la señal de control //
  if (u > umax) {
    u = umax;
  }
  else if (u < umin) {
    u = umin;
  }
  else
    u = u;

  c = (300-(300*u / umax) ); // escalado de señal a microsegundos para el disparo
del triac

if ((c<=1))
  c=1;
if (c>300)
  c=300;

OCR1A = c;
 //* Actualización de variables para la proxima iteración
  errAnterior = error;
  tiempo4 = tiempo3;
  }


//***** INTERRUPCION DE LECTURA DEL ENCODER ********//
void encoder() {
  //Serial.println("encoder");

  if (PASOS == 0)
    tiempo1 = millis();

  PASOS ++;

  if (PASOS >= vuelta) {
    tiempo2 = millis();
    tiempoVuelta = (tiempo2 - tiempo1);
```

```
    velocidad =  (float)(vuelta * 1000 / tiempoVuelta); // rps
    PASOS = 0;
  }
  //*********************/////************************////************

}

//* Interrupción de detección de pasovelocidada por cero y disparo del triac *//
void cero() {

  TCCR1B = 0x04; // inicia el temporizador con divide por entrada de 256
  TCNT1 = 0; // reiniciar el temporizador - contar desde cero
}

//******************* Disparo del Triac  ***************************//

ISR (TIMER1_COMPA_vect) {// comparador coincide con
  digitalWrite (TRIAC, HIGH); // establece la puerta TRIAC en
  TCNT1 = 65536-8; // ancho de impulso de disparo 16 bits
}

ISR(TIMER1_OVF_vect){ //timer1 overflow
  digitalWrite(TRIAC,LOW); // apaga la puerta TRIAC
  TCCR1B = 0x00;              // deshabilita el cronómetro para detener los
desencadenantes involuntarios
}


//******************* Interrupción de paro ************************//

void STOP() {
  Serial.println("paro");
  paro = true;
  marcha = false;
  digitalWrite(TRIAC, LOW);
  u = 0;
  I = 0;
  consigna=0;
}
```

ANEXO 3: Esquemas electrónicos


5V
R5
4.7kΩ
R2
10kΩ
R1
160Ω
R3
10kΩ
Q1
TLP621
C 470pF
R4
6.8kΩ
ENCODER
Arduino Mega
Rp
10kΩ
Rz
66kΩ
TLP621
FOD 814
230 V
MOC3031-M
R2
180Ω
R3
2kΩ
BTA 06 T405-T
R1
330Ω
MOC 3023
R4
39Ω
C1
0.1µF
C2
10nF
M1
MARCHA
C4
10nF
R6
4.7kΩ
PARO
C3
10nF
R5
4.7kΩ
Arduino
Mega
2560
(Rev3)
fritzing